Express Mail Label No. EV397262288US

[0001] # METHOD AND SYSTEM FOR STORING DATA USING A CONTINUOUS DATA PROTECTION SYSTEM

[0002] ## CROSS REFERENCE TO RELATED APPLICATION(S)

[0003] This application claims priority from U.S. Provisional Application No. 60/---,---, entitled "METHOD AND SYSTEM FOR CONTINUOUS DATA PROTECTION," filed on February 4, 2004, which is incorporated by reference as if fully set forth herein.

[0004] ## FIELD OF THE INVENTION

[0005] The present invention relates generally to continuous data protection, and more particularly, to storing data using a continuous data protection system.

[0006] ## BACKGROUND

[0007] Hardware redundancy schemes have traditionally been used in enterprise environments to protect against component failures. Redundant arrays of independent disks (RAID) have been implemented successfully to assure continued access to data even in the event of one or more media failures (depending on the RAID Level). Unfortunately, hardware redundancy schemes are very ineffective in dealing with logical data loss or corruption. For example, an accidental file deletion or virus infection is automatically replicated to all of the redundant hardware components and can neither be prevented nor recovered from by such technologies. To overcome this problem, backup technologies have traditionally been deployed to retain multiple versions of a production system over time. This allowed administrators to restore previous versions of data and to recover from data corruption.

[0008] Backup copies are generally policy-based, are tied to a periodic schedule, and reflect the state of a primary volume (i.e. a protected volume) at the particular point in time that is captured. Because backups are not made on a continuous basis, there will be some data loss during the restoration, resulting from a gap between the time when the backup was performed and the restore point that is required. This gap can be significant in typical environments where backups are only performed once per day. In a mission-critical setting, such a data loss can be catastrophic. Beyond the potential data loss, restoring a primary volume from a backup system, can be very complicated and often takes many hours to complete. This additional downtime further exacerbates the problems associated with a logical data loss.

[0009] The traditional process of backing up data to tape media is time driven and time dependent. That is, a backup process typically is run at regular intervals and covers a certain period of time. For example, a full system backup may be run once a week on a weekend, and incremental backups may be run every weekday during an overnight backup window that starts after the close of business and ends before the next business day. These individual backups are then saved for a predetermined period of time, according to a retention policy. In order to conserve tape media and storage space, older backups are gradually faded out and replaced by newer backups. Further to the above example, after a full weekly backup is completed, the daily incremental backups for the preceding week may be discarded, and each weekly backup may be maintained for a few months, to be replaced by monthly backups over time. It is noted that the daily backups would typically not all get discarded on the same day. Instead, the Monday backup set is overwritten on Monday, the Tuesday set is overwritten on Tuesday, etc. This ensures that a backup set is available that is within 8 business hours of any corruption that may have occurred in the past week.

[0010] Despite frequent hardware failures and the necessity of ongoing maintenance and tuning, the backup creation process can be automated, while restoring data from a backup remains a manual and time-critical process. First, the appropriate backup tapes need to be located, including the latest full backup and any incremental backups made since the last full backup. In the event that only a partial restoration is required, locating the appropriate backup tape can take just as long. Once the backup tapes are located, they must be restored to the primary volume. Even under the best of circumstances, this type of backup and restore process cannot guarantee high availability of data.

[0011] Another type of data protection involves creating point in time (PIT) copies of data. A first type of PIT copy is a hardware-based PIT copy, which is a mirror of the primary volume onto a secondary volume. The main drawbacks to a hardware-based PIT copy are that the data ages quickly and that each copy takes up as much disk space as the primary volume. A software-based PIT, typically called a "snapshot," is a "picture" of a volume at the block level or a file system at the operating system level. Various types of software-based PITs exist, and most are tied to a particular platform, operating system, or file system. These snapshots also have drawbacks, including occupying additional space on the primary volume, rapid aging, and possible dependencies on data stored on the primary volume wherein data corruption on the primary volume leads to corruption of the snapshot. In addition, snapshot systems generally do not offer the flexibility in scheduling and expiring snapshots that backup software provides.

[0012] While both hardware-based and software-based PIT techniques reduce the dependency on the backup window, they still require the traditional tape-based backup and restore process to move data from disk to tape media and to manage the different versions of data. This dependency on legacy backup applications and processes is a significant drawback of these technologies. Furthermore, like traditional tape-based backup and restore processes, PIT copies are made at discrete moments in time, thereby limiting any restores that are performed to the points in time at which PIT copies have been made.

[0013] A need therefore exists for a system that combines the advantages of tape-based systems with the advantages of snapshot systems and eliminates the limitations described above.

[0014] SUMMARY

[0015] The present invention is a method and system where data is structured so that writes may be continuously duplicated and a protected volume may be restored to any particular point in time, as desired. Writes are continuously logged to a secondary volume in a sequential fashion and meta-data regarding the secondary volume is organized in the form of delta maps. The delta maps indicate which volume blocks were written to during a particular time frame and point to the location of the actual modified data blocks that were written during this time frame.

[0016] BRIEF DESCRIPTION OF THE DRAWING(S)

[0017] A more detailed understanding of the invention may be had from the following description of a preferred embodiment, given by way of example, and to be understood in conjunction with the accompanying drawings wherein:

[0018] Figures 1A-1C are block diagrams showing a continuous data protection environment in accordance with the present invention;

[0019] Figure 2 is a delta map in accordance with the present invention.

[0020] Figure 3 is a hierarchy of delta maps for storing data in a continuous data protection system;

[0021] Figure 4 illustrates how delta maps are merged and how they are used to rewind a system to a previous point in time;

[0022] Figure 5 is a method wherein data is written to a continuous data protection system;

[0023] Figure 6 is a flow diagram illustrating the steps involved when a write is made by a primary volume; and

[0024] Figure 7 is an overview of the operation of the complete data protection system.

[0025] DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT(S)

[0026] In the present invention, data is backed up continuously, allowing system administrators to pause, rewind, and replay live enterprise data streams. This moves the traditional backup methodologies into a continuous background process in which policies automatically manage the lifecycle of many generations of restore images.

[0027] Figure 1A shows a preferred embodiment of a protected computer system 100 constructed in accordance with the present invention. A host computer 102 is connected directly to a primary data volume 104 (the primary data volume may also be referred to as the protected volume) and to a data protection system 106. The data protection system 106 manages a secondary data volume 108. The construction of the system 100 minimizes the lag time by writing directly to the primary data volume 104 and permits the data protection system 106 to focus exclusively on managing the secondary data volume 108. The management of the volumes is preferably performed using a volume manager. The volume manager mirrors writes so that one copy goes to the primary volume 108 and one copy goes to the data protection system 106 which in turn writes sequentially to the secondary volume 108. That is, the data protection pretends to be a regular disk/volume but when writes arrive for a specific address, the system stores them sequentially on the secondary volume 108. A volume manager is software module that runs on a server or intelligent storage switch to manage storage resources. Typical volume managers have the ability to aggregate blocks from multiple different physical disks into one or more virtual volume. Applications are not aware that they are actually writing to segments of many different disks because one large, contiguous volume is presented to them. In addition to block aggregation, volume managers usually also offer software RAID functionality. For example, they are able to split the segments of the different volumes into two groups, where one group is a mirror of the other group. This is, in a preferred embodiment, the feature the present data protection system is taking advantage of when it is implemented as shown in Figure 1A. In many environments, the volume manager or host-based driver already mirrors the writes to two distinct primary volumes for redundancy in case of a hardware failure. The present invention is configured as a tertiary mirror target in this scenario, such that the volume manager or host-based driver also send copies of all writes to the data protection system.

[0028] It is noted that the primary data volume 104 and the secondary data volume 108 can be any type of data storage, including, but not limited to, a single disk, a disk array (such as a RAID), or a storage area network (SAN). The main difference between the primary data volume 104 and the secondary data volume 108 lies in the structure of the data stored at each location, as will be explained in detail below. It is noted that there may also be differences in terms of the technologies that are used. The primary volume is typically an expensive, very fast, highly available storage subsystem, whereas the secondary volume is typically cost-effective, high capacity and comparatively slow (for example, ATA/SATA disks). Normally, the slower secondary volume cannot be used as a synchronous mirror to the high-performance primary volume. This is because the slow response time would have an adverse impact on the overall system performance. The disclosed data protection system, however, is optimized to keep up with high-performance primary volumes. These optimizations are described in more detail below. At a high level, random writes to the primary volume are processed sequentially on the secondary storage. Sequential writes improve both the cache behavior, as well as the actual volume performance of the secondary volume. In addition, it is possible to aggregate multiple sequential writes on the secondary volume, whereas this is not possible with the random writes to the primary volume. Also note that the present invention does not require writes to the data protection system to be synchronous. However, even in the case of an asynchronous mirror, minimizing latencies is important.

[0029] Figure 1B shows an alternate embodiment of a protected computer system 120 constructed in accordance with the present invention. The host computer 102 is directly connected to the data protection system 106, which manages both the primary data volume 104 and the secondary data volume 108. The system 120 is likely slower than the system 100 described above, because the data protection system 106 must manage both the primary data volume 104 and the secondary data volume 108. This results in a higher latency for writes to the primary volume in the system 120 and lowers the available bandwidth for use. Additionally, the introduction of a new component into the primary data path is undesirable because of reliability concerns. Nonetheless, it is a usable configuration for lower-end deployments.

[0030] Figure 1C shows another alternate embodiment of a protected computer system 140 constructed in accordance with the present invention. The host computer 102 is connected to an intelligent switch 142. The switch 142 is connected to the primary data volume 104 and the data protection system 106, which in turn manages the secondary data volume 108. The switch 142 includes the ability to host applications and contains some of the functionality of the data protection system 106 in hardware, to assist in reducing system latency and to improve bandwidth.

[0031] It is noted that the data protection system 106 operates in the same manner, regardless of the particular construction of the protected computer system 100, 120, 140. The major difference between these deployment options is the manner and place in which a copy of each write is obtained. To those skilled in the art it is evident that other embodiments, such as the cooperation between a switch platform and an external server, are also feasible.

[0032] In practice, certain applications require continuous data protection with a block-by-block granularity, for example, to rewind individual transactions. However, the period in which such fine granularity is required is, generally, relatively short (for example two days), which is why the system can be configured to fade out data over time. The present invention discloses data structures and methods to manage this process automatically.

[0033] Because data is continuously backed-up in the present invention, reversing each write to get to a particular point in time quickly becomes unfeasible where hundreds, thousands or more writes are logged every second. The amount of data simply becomes too large to scan in a linear fashion. The present invention therefore provides data structures (i.e. delta maps) so that such voluminous amounts of backup data may be efficiently tracked and accessed, as desired.

[0034] In typical recovery scenarios, it is necessary to examine how the primary volume looked like at multiple points in time before deciding which point to recover to. For example, consider a system that was infected by a virus. In order to recover from this virus, it is necessary to examine the primary volume as it was at different points in time in order to find the latest recovery point where the system was not yet infected by the virus. In order to efficiently compare multiple potential recovery points, additional data structures are needed. Delta maps provide a mechanism to efficiently recover the primary volume as it was at a particular point in time, without the need to replay the write log in its entirety, one write at a time. In particular, delta maps are data structures that keep track of data changes between two points in time. These data structures can then be used to selectively play back portions of the write log such that the resulting point-in-time image is the same as if the log were played back one write at a time, starting at the beginning of the log.

[0035] Referring now to Figure 2, there is shown a delta map 150 in accordance with the present invention. While the format shown in Figure 2 is preferred, any format may of course be used. For each write to a primary volume, a duplicate write is made, in sequential order, to a secondary volume. To create a mapping between the two volumes, it is preferable to have an originating entry and a terminating entry for each write. The originating entry includes information regarding the origination of a write. The terminating entry includes information regarding the termination of a write. For example, as shown in delta map 150, row 160 is an originating entry and row 162 is a terminating entry. Row 160 includes a field 152 for specifying the region of a primary volume wherein the first block was written to; a field 154 for specifying the block offset within that region; a field 156 for specifying where on the secondary volume the duplicate write (i.e. the copy of the primary volume write) begins; and a field 158 for specifying the physical device used to make the write persistent. Row 162 includes a field 164 for specifying the region of the primary volume wherein the last block was written to; a field 166 for specifying the block offset in a region of a primary volume at which the write ends; and a field 168 for specifying where on the secondary volume the duplicate write ends or alternatively, where the next write starts. While field 170 is provided in a terminating entry such as row 162, it is noted that such a field 170 is not necessary since there is no physical device usage associated with termination of a write. Also note that rows 160 and 162 follow the same format in this implementation. In other words, the terminating entry conforms to the same notation as the initiating entry. This is intentional because if a new block were written consecutive to the first block, the initiating entry of the second block would also be the terminating entry for the first block, thus guaranteeing a very compact and space efficient notation.

[0036] In a preferred embodiment, as explained above, each delta map contains a list of all blocks that were changed during the particular time period to which the delta map corresponds. That is, each delta map specifies a block region on the primary volume, the offset on the primary volume, and physical device information. This information can then be used to recreate the primary volume as it looked like at a previous point in time. For example, assume that a volume was brand new and that only the two writes in delta map 150 have been committed to it. The map thus contains a list of all modifications since the volume was in its original state. In order to recreate the volume as it was after these two writes (for example, after a failure of the primary volume), the system examines the first two entries in the delta map. These entries are sufficient to determine that a block of data had been written to region R0 on the primary disk at offset 100 and that the length of this write was 20. In addition, fields 156 and 158 can be used to determine where the duplicate copy was written on the secondary volume. This process can then be repeated for each entry and an exact copy of the primary volume at that time can be recreated in this fashion. It is noted that other fields or a completely different mapping format may be used while still achieving the same functionality. For example, instead of dividing the primary volume into block regions, a bitmap could be kept, representing every block on the primary volume.

[0037] Referring now to Figure 3, there is shown a hierarchy 200 of delta maps for storing data in a continuous data protection system. Delta maps 202 are initially created from the write log, using a map engine. This could occur in real-time or after a certain number of writes or according to a time interval, etc. Additional delta maps may also be created as a result of a merge process. Such maps are referred to as merged delta maps 204 and 206 and may be created to optimize the access and restore process. The delta maps 202 are stored on the secondary volume and contain a mapping of the primary address space to the secondary address space. The mapping is kept in sorted order based on the primary address space as illustrated in Figure 2.

[0038] As explained above, each delta map includes information necessary to recreate the changes to the protected volume for a particular time window. For example, delta map $202_{t0\text{-}t1}$ corresponds to the change of the protected volume between time zero and time one, delta map $202_{t1\text{-}t2}$ corresponds to the change of the protected volume from time one to time two, and so forth. It is noted that these time windows do not necessarily need to be of equal size. If a primary volume is completely destroyed at time n+1, a full restore as of time n may be performed by simply using merged delta map $206_{t0\text{-}tn}$. If a loss occurs at time three, and the primary volume needs to be restored, merged delta map $202_{t0\text{-}t3}$ may be used. If a loss occurs at time five and the system needs to be restored to time four, merged delta map $204_{t0\text{-}t3}$ and delta map $204_{t3\text{-}t4}$ may be used.

[0039] As shown in Figure 3, the delta maps are chained together in chronological order. In the preferred embodiment, each delta map points to the previous delta map, a merged delta map if it exists, and, in the case of a pre-merged delta map, to the first child delta map. Other embodiments are possible, for example, where both forward and backward pointers are kept or only forward pointers are kept. The protected volume may therefore be restored using any delta map or pre-merged delta map as desired. In Figure 3, three levels of pre-merging are implemented. However, it is possible to create a hierarchy of fewer or more levels, as desired.

[0040] Referring now to Figure 4, two delta maps 450 and 452 and a merged delta map 454 are shown. In Figure 4, four writes 456, 458, 460, 462 have been written to a primary volume 464. Increasing reference numerals are used to denote the point in time at which each write occurred. That is, write 456 was before write 458 and so forth. As can be seen from comparing the primary and secondary volumes 464, 466, each write made to the primary volume 464 is duplicated in sequential fashion to the secondary volume 466. Duplicating the writes in sequential fashion allows the order in which writes occurred on the primary volume to be readily apparent from the layout of the secondary volume. Conversely, the original location is not apparent and needs to be stored in the delta maps, as discussed. This example illustrates the difference between block-order and time-order.

[0041] Delta map 450 includes the originating and terminating entries for writes 456 and 458 while delta map 452 includes originating and terminating entries for writes 460 and 462. In delta map 450, the two top entries are the originating and terminating entries for write 456 and the two bottom entries are the originating and terminating entries for write 458. Similarly, the two top entries in delta map 452 are the originating and terminating entries for write 460 and the two bottom entries are the originating and terminating entries for write 462. As explained above, the delta maps 450 and 452 include the specifics regarding each write that occurred during the time period covered by the particular delta map.

[0042] Delta maps 450 and 452 may be merged into a single merged map 454. One significant benefit of merging delta maps is a reduction in the number of entries that are required. Another, even more significant benefit, is a reduction in the number of blocks that need to be kept on the secondary volume once the lower-level maps are expired. It is noted, however, that this is only the case when a previous block was overwritten by a newer one. For example, in this particular scenario, it is possible to eliminate the terminating entry 468 of write 462 because writes 462 and 458 are adjacent to each other on the primary volume. That is, because there is a terminating entry 468 with the same offset (i.e. 240) as an originating offset 470, the terminating entry 468 may be eliminated in merged delta map 454. By way of further example, if a subsequent write was performed that entirely filled region two (i.e. R2), and the map containing that write was merged with map 454, all of the entries related to R2 would be replaced with the R2 originating and terminating entries for the subsequent write. In this case, it will also be possible to free up the blocks in this region once the delta maps are expired. The delta maps and the structures created by merging maps reduces the amount of overhead in maintaining the mapping between the primary and secondary volumes over time.

[0043] Referring now to Figure 5, there is shown a method wherein data is written to a continuous data protection system. In this implementation, the primary and secondary storage are managed by the same software, which could either be a host-based driver/software or a switch/appliance. First, it is determined in step 302 whether the system is writing to a clean region. If yes, the dirty region log is updated in step 304 and the system writes to the primary and secondary volumes in step 306. If no, the method 300 proceeds directly from step 302 to step 306.

[0044] If writing to the primary volume is complete (step 308), the method proceeds to step 310 wherein the status (i.e., an indication that the primary volume write is complete) is sent to the host. It is important to note that a "good" status can be returned without regard to whether the data made it to the secondary volume in this embodiment. This is advantageous for performance reasons. However, a synchronous embodiment is also possible as described above. The method 300 then proceeds to step 312 to check for errors in the primary volume write. If an error has occurred, an additional entry is added to the write log in step 314 reflecting the fact that an error has occurred and then the method 300 proceeds to step 316. If no error has occurred, the method 300 proceeds directly from step 312 to step 316. In step 316, it is confirmed whether writing to the secondary volume is complete. Once the write is completed, the method 300 proceeds to step 318 to check for errors in the secondary volume write. If an error has occurred, an entry reflecting the fact that an error has occurred is added to the write log (step 320).

[0045] By way of further explanation, the sequence of events performed when a host computer performs a write to a primary volume is shown in Figure 6. When a host performs a write 350, an entry is added to a write log buffer 352. It is noted that if the write log buffer is full, it is asynchronously flushed. When a write log buffer is flushed, the writes are written out to a secondary volume. That is, duplicated writes may be kept in a buffer and written to the secondary volume whenever the buffer gets full or at another convenient time. Non-volatile RAM can be used to increase this buffer safely.

[0046] If the write is to a clean region, a synchronous update of the dirty region log (DRL) is performed 354. Then, both the primary and secondary writes are started 356, 358. Once the primary write is completed 360, the status is returned to the host 362. If a host-based volume manager is used, this happens independently of the secondary write. If an error occurred in the primary write, it is indicated in the write log by adding an additional entry. Of course, there are two possibilities with respect to the completion of the secondary write (i.e. the duplicate write made to a secondary volume). That is, the secondary write may be completed before 364 or after 366 the completion of the primary write 360. It is noted that whether the secondary write is completed before 364 or after 366 does not affect implementation of the present invention. As with completion of the primary write, if an error occurred in the secondary write, it is indicated in a write log by adding an additional entry to the write log.

[0047] Referring now to Figure 7, the data protection system preferably operates according to method 400. The method 400 begins with step 402 wherein an initial full copy snapshot is created. The initial snapshot is the first snapshot taken of the data existing on the primary volume.

[0048] By way of explanation, assuming the present invention is implemented to protect a system that is currently in production. Such a system already contains important data on the primary volume at time t0. Hence, if the system starts recording the changes from this point on, the volume cannot be reconstructed at a later time unless there is a copy of the volume as it was at time t0. The initial full copy does exactly this – it initializes the secondary volume to a state where the contents of each block at time t0 are known. It is noted that in the special case when the primary volume is empty or will be formatted anyway, users have the option to disable the initial full copy. This means, however, that if they want to restore the volume back to time t0, an empty volume will be presented. To provide further explanation, assume that the primary volume already contains important data. In that case, if blocks 5, 9, and 57 are overwritten at times t1, t2, and t3 respectively, it is not possible to present a complete volume image as it was at time t2. This is because it is not known what blocks 1, 2, 3, 4, 6, 7, 8, 10, 11, etc. looked like at that time without having taken a full copy snapshot first.

[0049] Referring again to Figure 7, the initial snapshot establishes time zero for the secondary volume, as explained above. Once the state of the primary volume at time zero is established, each write to the primary volume is split and duplicated to the secondary volume (step 404). In step 406, it is determined whether a snapshot is triggered. If so, a marker is inserted into the secondary volume write log in step 408 and then the method proceeds to step 410. A snapshot in this context is simply a point in time to which the system can recover to at a later time. Delta maps are kept for that point in time until the snapshot expires. Note that because all data has already been stored in the write log, no data movement is necessary to take a snapshot. If no snapshot is taken, the method 400 proceeds directly to step 410.

[0050] In step 410, a delta map is created by converting the time-ordered write log entries to a block-ordered delta map. Next, in step 412, it is determined whether pre-merge optimization will be performed. If so, delta maps are periodically merged to provide a greater granularity in the data on the secondary volume. It is noted that merging of delta maps can occur at any time and according to any desired policy. In the preferred embodiment, the merging algorithm looks for adjacent delta maps with the same expiration policy and merges those. This minimizes the number of merge operations that will be required upon expiration. In a different embodiment, pre-merging occurs automatically after a certain number of writes W or after a certain time period T. Additionally, the system is capable of storing full maps of the primary volume at various points in time. This significantly accelerates the merging process later because fewer maps need to be merged. Regardless of whether pre-merge optimization is performed, the method 400 cycles back to step 404.

[0051] Although the present invention has been described in detail, it is to be understood that the invention is not limited thereto, and that various changes can be made therein without departing from the spirit and scope of the invention, which is defined by the attached claims.

* * *